Taller de Sistemas Operativos

Linux Programming Interface

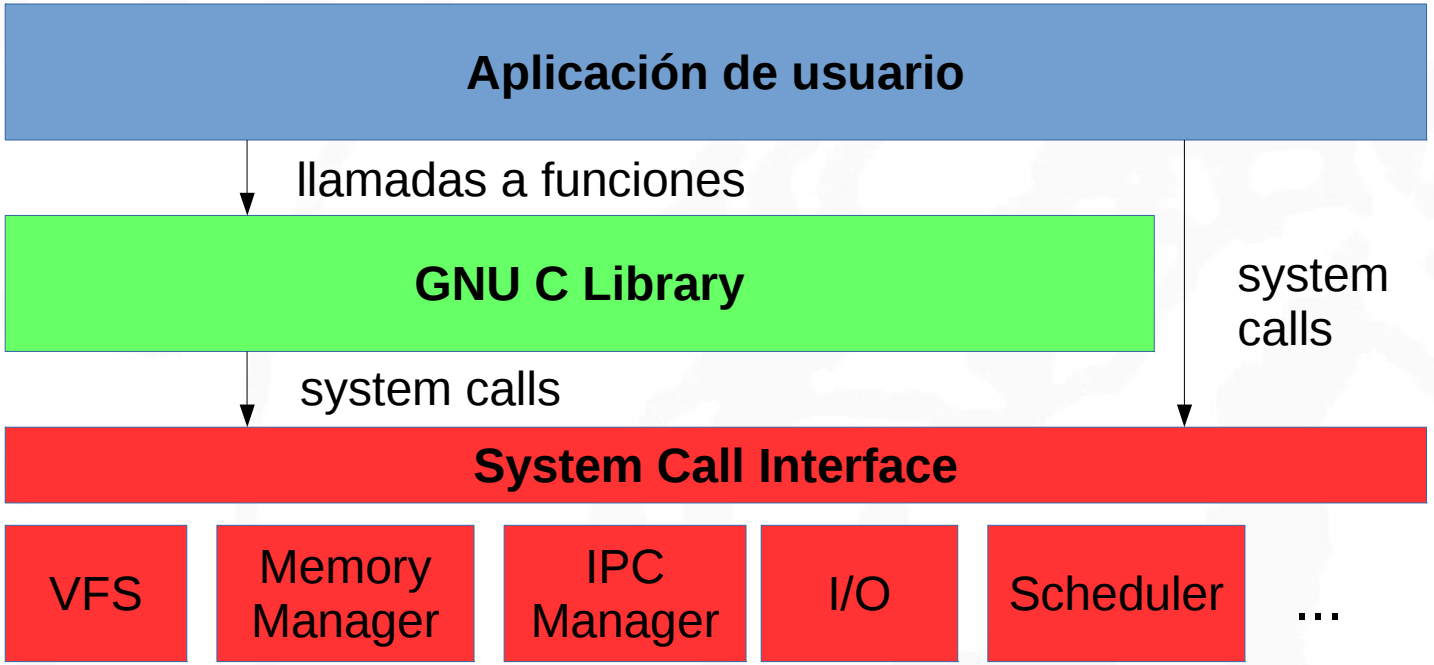
Inter Process Communication (IPC)

Agenda

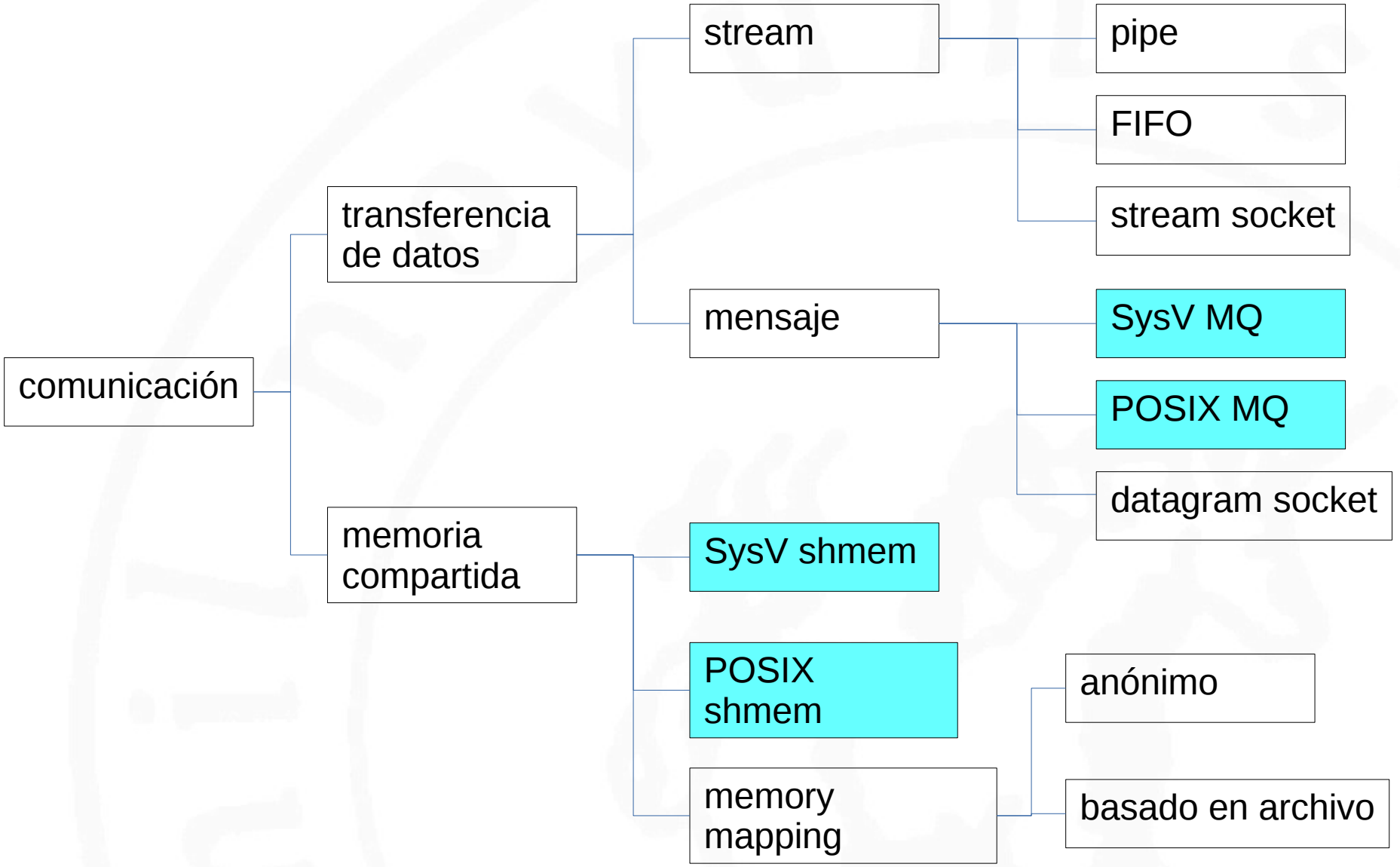
- Introducción
- Colas de mensajes
- Memoria compartida
- Semáforos
  - Implementación en el kernel

GNU C Library (glibc)

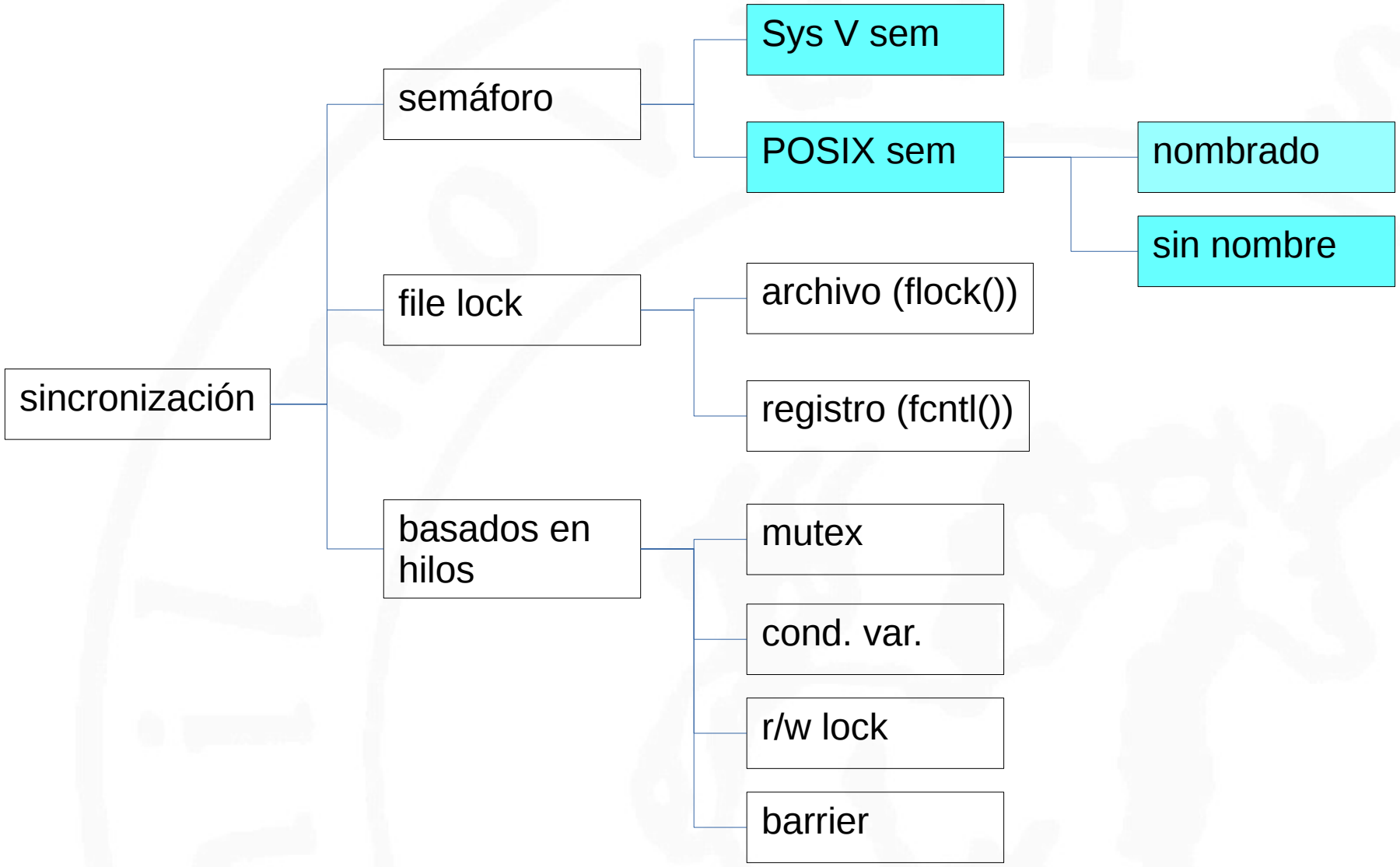
- Las aplicaciones de usuario no llaman directamente al kernel (normalmente)
  - Utilizan funciones wrapper de la bilblioteca de C
- La implementación de la bilbioteca de C más usada en Linux es GNU C library o glibc



Comunicación



Sincronización



POSIX vs Sys V

- System V Interface Definition (Sys V)
  - Basado en AT&T Unix System V
  - Primera versión publicada en 1985
  - Última versión publicada en 1995
- Portable Operating System Interface (POSIX)
  - Especificado por IEEE Computer Society
    - Independiente de vendedor
  - Primera versión publicada en 1988
    - En parte basada en Sys V
  - Última versión publicada en 2013

Taller de Sistemas Operativos

Sys V IPC

Sys V

- Comandos generales
  - ipcs: información de los recursos IPC del sistema
  - ipcrm: elimina un recurso IPC
  - ipcmk: crea un recurso IPC
- Sys V usa claves de tipo key_t para identificar recursos
  - key_t ftok(char *pathname, char proj);

Sys V - MQ

- Comunicación orientada a mensajes
  - El receptor lee un mensaje por vez
  - Nunca se leen mensajes parciales ni múltiples
  - Los mensajes tienen tipo y se pueden filtrar por tipo

Sys V - MQ

- int msqid = msgget (key_t key, int msgflg);
  - key: resultado de invocación a ftok()
  - msgflg: bandera de opciones
    - IPC_CREAT : crear MQ si no existe
    - IPC_EXCL : crear MQ exclusivamente
    - 0ddd: especifica los permisos a asignar sobre el recurso en octal.
- msqid es el identificador de recurso si hubo éxito.

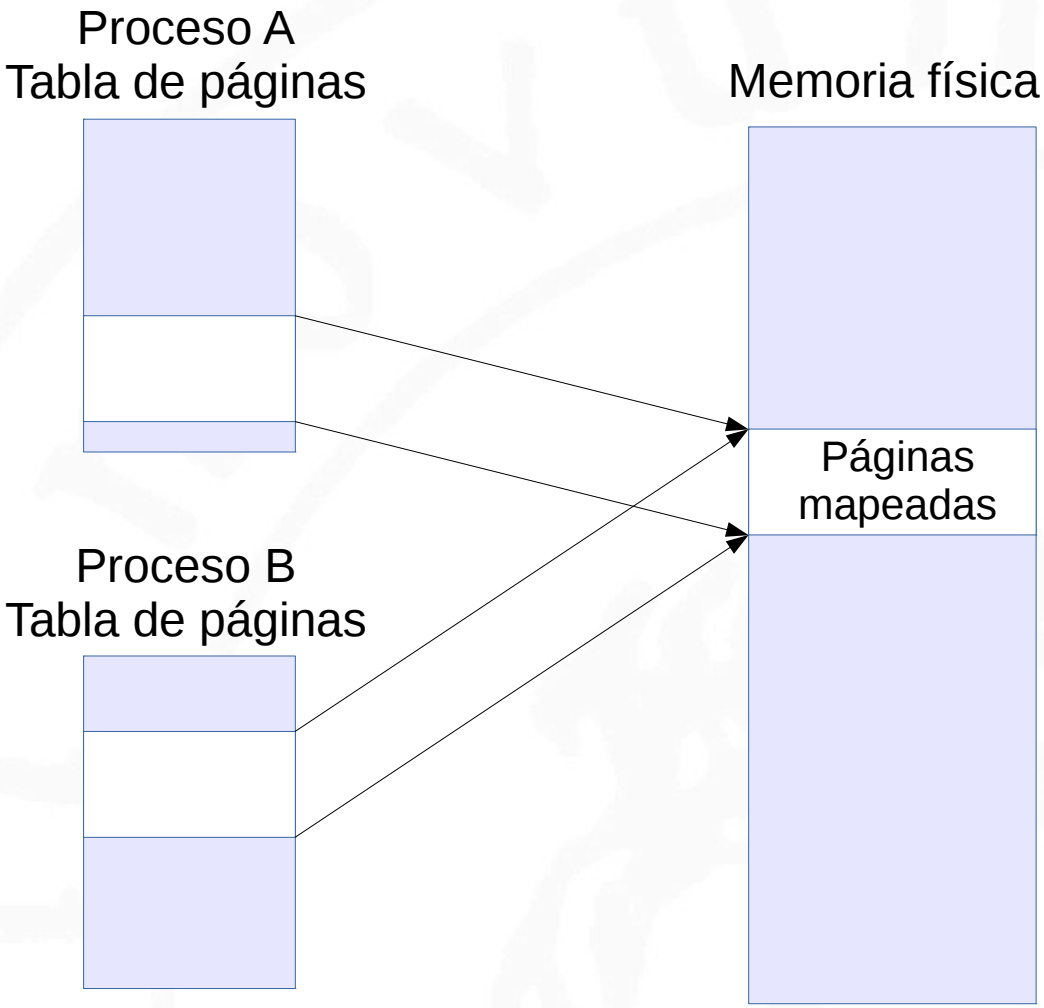
Sys V - MQ

- Emvía un mensaje
  - msgsnd (msqid, *msgp, msgsz, msgflg);
- Recibe un mensaje
  - msgrcv(msqid, *msgp, msgsz, mtype, msgflg);
- Administración de una MQ
  - msgctl(msqid, cmd, struct msqid_ds *buf);

Sys V - shm

- Permite que múltiples procesos compartan la misma página física de memoria
- Comunicación – copiar datos a memoria
- Muy eficiente
  - Transferencia de datos (e.g. MQ):
    - espacio de usuario → kernel → espacio de usuario
  - Memoria compartida:
    - copia simple en espacio de usuario
- Pero requiere sincronización para su acceso

Sys V - shm


Proceso A
Tabla de páginas
Memoria física
Páginas
mapeadas
Proceso B
Tabla de páginas

Sys V - shm

- int shmid = shmget (key_t key, int size, int shmflg);
  - key: resultado de invocación a ftok()
  - size: tamaño del segmento solicitado.
  - shmflg: bandera de opciones
    - IPC_CREAT: crea el segmento si no existe
    - IPC_EXCL: crea de forma exclusiva
    - 0ddd: especifica los permisos a asignar sobre el segmento en octal.
  - shmid: identifica el segmento de memoria compartida

Sys V - shm

- void *addr = shmat (int shmid, void *shmaddr, int shmflg);
  - shmid: identificador resultado de shmget()
  - shmaddr: cero para que el sistema operativo ubique libremente el segmento
  - shmflg: bandera de opciones
    - SHM_RDONLY : para indicar sólo lectura.
- *addr dirección de la memoria compartida

Sys V - shm

- Administrar un segmento de memoria compartida
  - shmctl (shmid, cmd, struct shmid_ds *buf);
- Liberar un segmento de memoria compartida
  - shmdt (*shmaddr);
- Para borrar:
  - shmctl (shmid, IPC_RMID, ...)

Sys V - sem

- Mantiene un valor entero
- Permite modificar arbitrariamente el valor del semáforo
  - Sumar o restar un valor N al semáforo
  - El proceso se bloquea si resta un valor mayor al valor actual
- Permite bloquear un proceso hasta que el semáforo tenga un valor 0
- Un id proporciona acceso a un array de semáforos
  - Podemos aplicar operaciones grupales

Sys V - sem

- int semid = semget (key_t key, int nsem, int semflg);
  - key: resultado de invocación a ftok()
  - nsem: cantidad de semáforos en el recurso.
  - semflg: bandera de opciones
    - IPC_CREAT: crea el segmento si no existe
    - IPC_EXCL: crea de forma exclusiva
    - 0ddd: especifica los permisos a asignar sobre el recurso en octal.
- semid identifica al semáforo creado

Sys V - sem

- Operaciones sobre los semáforos del array
  - int semop (semid, sembuf *sops, nsops);

```
struct sembuf:
ushort  sem_num; // índice del semáforo
short   sem_op;  // operación
short   sem_flg;
```

- Administrar semáforo
  - int semctl (semid, semnum, cmd, arg);

Taller de Sistemas Operativos

POSIX IPC

POSIX MQ

- Comunicación orientada a mensajes
  - El receptor lee un mensaje por vez
  - Nunca se leen mensajes parciales ni múltiples
- Los mensajes tienen prioridad
  - Son entregados en orden de prioridad

POSIX MQ - API

- Creación/destrucción
  - mq_open(): abre/crea una MQ
  - mq_close(): cierra una MQ
  - mq_unlink(): elimina un MQ pathname
- I/O
  - mq_send(): envía un mensaje
  - mq_receive(): recibe un mensaje
- Otros
  - mq_setattr(), mq_getattr(): set/get atributos
  - mq_notify(): solicita notificación de arribo de mensaje

POSIX MQ - mq_open()

- mqd = mq_open(nombre, flags, [modo, &atributos]);
- Crea una nueva MQ o abre una MQ existente
  - nombre: tiene la forma “/algunnombre”
    - Visible en /dev/mqueue
  - mpd: es de tipo struct mqd_t

POSIX MQ - mq_open()

- mqd = mq_open(nombre, flags, [modo, &atributos]);
  - flags: (análogo a open())
    - O_CREAT – crea un MQ si no existe
    - O_EXCL – crea un MQ exclusivamente
    - O_RDONLY, O_WRONLY, O_RDWR
    - O_NONBLOCK – I/O no bloqueante
  - modo: setea permisos
  - &attr: setea atributos para la MQ
    - NULL indica atributos por defecto

POSIX MQ - mq_open()

- Ejemplos:

```
// Crea una MQ de forma exclusiva y
// la abre para escritura
mqd = mq_open(“/mq_tso”, O_CREAT| O_EXCL
  O_WRONLY, 0600, NULL);

// Abre una MQ existente para lectura
mqd = mq_open(“/mq_tso”, O_RDONLY);
```

POSIX MQ - mq_unlink()

- mq_unlink(nombre);
  - Elimina una la MQ identificada por nombre y la marca para ser destruida
  - Es efectivamente destruida cuando todos los procesos dejan de usarla (mq_close)

POSIX MQ – I/O no bloqueante

- MQ tienen una capacidad máxima de mensajes
- Por defecto:
  - mq_receive(): bloquea si no hay mensajes en la MQ
  - mq_send(): se bloquea si la MQ está llena
- O_NONBLOCK
  - Retorna error EAGAIN en lugar de bloquearse

POSIX MQ – mq_send()

- mq_send(mqd, msg_ptr, msg_len, msgprio);
  - mqd: descriptor de MQ
  - msg_ptr: puntero al mensaje
  - msg_len: tamaño del mensaje
  - msgprio: prioridad del mensaje
    - entero positivo
    - 0 es la prioridad más baja

POSIX MQ – mq_send()

- Ejemplo:

```
mqd_t mpd;

mqd = mq_open("/mq_tso",O_CREAT |
  O_WRONLY, 0600, NULL);

char *msg = "hola mundo!";

mq_send(mqd, msg, strlen(msg), 0);
```

POSIX MQ - mq_receive()

- nb = mq_receive(mqd, msg_ptr, msg_len, &prio);
  - mqd: descriptor de MQ
  - msg_ptr: puntero a buffer de recepción
  - msg_len: tamaño del buffer
  - &prio: prioridad del mensaje recibido
  - nb: tamaño del mensaje recibido (bytes)

POSIX MQ - mq_receive()

- Ejemplo:

```
const int BUF_SIZE = 1000;
char buf[BUF_SIZE];
unsigned int prio;

mqd = mq_open("/mq_tso", O_RDONLY);
nbytes = mq_receive(mqd, buf, BUF_LEN,
  &prio);
```

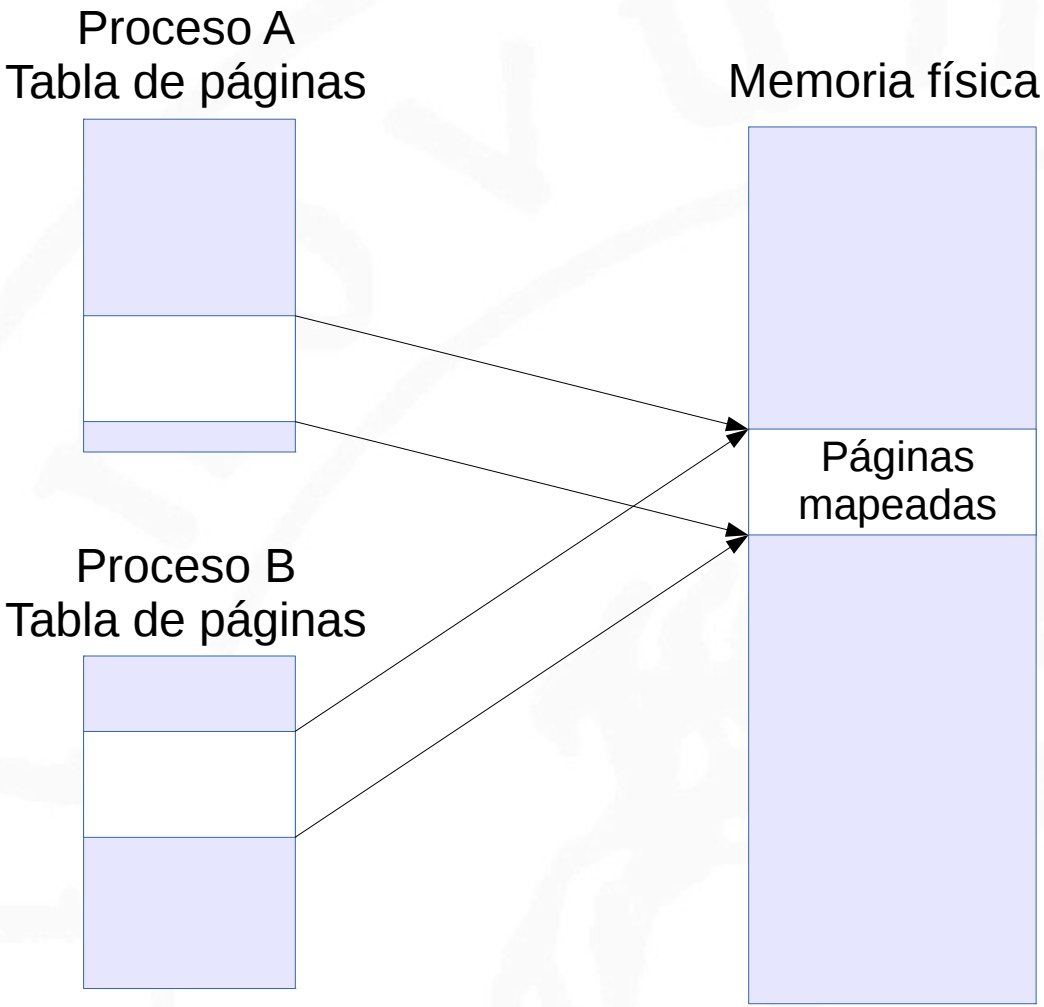
POSIX MQ - mq_notify()

- mq_notify(mqd, evento);
- Solamente un proceso puede registrarse
- Notifica cuando un mensaje llega a una cola vacía
  - Solamente si ningún otro proceso está ejecutando mq_receive()
- evento indica cómo será notificado el proceso
  - SIGEV_SIGNAL: el proceso recibe un signal
  - SIGEV_THREAD: nuevo hilo
- Funciona una única vez
  - Debe ser re-habilitado cada vez

POSIX mmap

- Permite que múltiples procesos compartan la misma página física de memoria
- Comunicación – copiar datos a memoria
- Muy eficiente
  - Transferencia de datos (e.g. MQ):
    - espacio de usuario → kernel → espacio de usuario
  - Memoria compartida:
    - copia simple en espacio de usuario
- Pero requiere sincronización para su acceso

POSIX mmap


Proceso A
Tabla de páginas
Memoria física
Páginas
mapeadas
Proceso B
Tabla de páginas

POSIX mmap - API

- addr = mmap(daddr, len, prot, flags, fd, offset);
  - daddr: ubicación del mapeo
    - NULL para que el kernel decida
  - len: tamaño del mapeo
  - prot: permisos sobre la memoria
    - PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE
  - flags: comportamiento
    - MAP_SHARED, MAP_ANONYMOUS
  - fd: file descriptor para el mapeo
  - offset: desplazamiento del comienzo de mapeo
  - addr: retorna la dirección usada para el mapeo

POSIX mmap

- Ejemplo: mapea un SHM de tamaño SIZE bytes.

```
addr = mmap(NULL, SIZE, PROT_READ |
  PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS,
  -1, 0);

pid = fork();
```

POSIX shmem - API

- Creación/destrucción
  - shm_open(): abrir/crear SHM
  - mmap(): mapea SHM
  - shm_unlink(): elimina un SHM
- Gestión de un SHM usando file descriptor
  - fstat(): retorna info (tamaño, dueño, permisos)
  - ftruncate():cambia el tamaño
  - fchown(),fchmod(): cambia dueño/permisos

POSIX shmem - shm_open()

- fd = shm_open(nombre, flags, modo);
- Abre/crea un objeto SHM
- nombre tiene la forma “/algunnombre”
  - Visible en /dev/shm
- Retorna un file descriptor: fd

POSIX shmem - shm_open()

- fd = shm_open(nombre, flags, modo);
  - flags: (análogo a open()):
    - O_CREAT – crea un SHM si no existe
    - O_EXCL – crea un SHM exclusivamente
    - O_RDONLY, O_RDWR
    - O_TRUNC – trunca un SHM existente a 0 bytes
  - modo: setea permisos

POSIX shmem - shm_open()

- Ejemplo: crea y mapea un SHM de tamaño SIZE bytes.

```
fd = shm_open(“/shm_tso”, O_CREAT | O_EXCL
  | O_RDWR, 0600);

// Establece el tamaño
ftruncate(fd, SIZE);

addr = mmap(NULL, SIZE, PROT_READ |
  PROT_WRITE, MAP_SHARED, fd, 0);
```

POSIX shmem - shm_open()

- Ejemplo: abre y mapea un SHM de tamaño desconocido.

```
fd = shm_open(“/shm_tso”, O_RDWR, 0);

// Obtiene el tamaño
struct stat sb;
fstat(fd, &sb);

addr = mmap(NULL, sb.st_size, PROT_READ |
  PROT_WRITE, MAP_SHARED, fd, 0);
```

POSIX sem

- Mantiene un valor entero
- No soporta arreglo de semáforos
- Solamente dos operaciones fundamentales
  - sem_post(): incrementa valor en 1
  - sem_wait(): decrementa valor en 1
    - Bloquea el proceso si valor < 0
- Dos tipos de semáforos
  - Sin nombre
    - Embebidos en memoria compartida
  - Nombrados
    - Objetos independientes

POSIX sem – sin nombre

- sem_init(s, pshared, value): inicializa el semáforo s con valor value
  - sem_t *s
  - pshared: 0 compartido entre hilos; !=0 compartido entre procesos
- sem_post(s): suma 1 al valor de s
- sem_wait(s): resta 1 al valor de s
- sem_destroy(s): libera el semáforo
  - ¡No debe haber ningún proceso esperando!

POSIX sem – sin nombre

- Ejemplo muy simple:

```
sem_t *s =(sem_t*)mmap(NULL, sizeof(struct
  sem_t), PROT_READ | PROT_WRITE,
  MAP_SHARED | MAP_ANONYMOUS, -1, 0);

sem_init(s,1,0);

if (fork() == 0) {
  sem_wait(s);
} else {
  sem_post(s);
}
```

POSIX sem – nombrados

- sem_open(): abre/crea un semáforo
- sem_unlink(): elimina un semáforo

POSIX sem – nombrados

- semp = sem_open(nombre, flags [, modo, valor]);
  - Abre/crea un semáforo
  - nombre tiene la forma “/algunnombre”
    - Visible en /dev/shm
    - Con nombre “sem.algunnombre”
  - Retorna sem_t *semp

POSIX sem – nombrados

- semp = sem_open(nombre, flags [, modo, valor]);
  - flags: (análogo a open()):
    - O_CREAT – crea un SEM si no existe
    - O_EXCL – crea un SEM exclusivamente
  - modo: setea permisos
  - valor: inicializa el semáforo

Implementación POSIX sem

- Mecanismos tradicionales (e.g. Sys V)
  - Locking a nivel de kernel
  - El kernel provee de forma natural el estado compartido y maneja la concurrencia
  - Todos los accesos requieren una system call

Implementación POSIX sem

- Linux modernos usan Fast Userlevel Mutex (FUTEX) para la sincronización fuera del kernel
  - Locking a nivel de usuario
  - Operaciones atómicas sobre el lock
  - El kernel interviene solo en casos competidos
    - futex syscall

Implementación POSIX sem - userlevel

```
struct sem_t
{
  // Valor del semáforo
  unsigned int value;
  // Cantidad de procesos en espera
  unsigned long int nwaiters;
  int private;
};
```

Implementación POSIX sem - userlevel

```
int sem_wait(sem_t *sem) {
    if (atomic_decrement_if_positive(&sem->value))
      return 0;
    atomic_increment(&sem->nwaiters);
    for (;;) {
      sys_futex(sem, FUTEX_WAIT, 0);
      if (atomic_decrement_if_positive(
                                        &sem->value))
        break;
    }
    atomic_decrement(&sem->nwaiters);
    return 0;
}
```

Implementación POSIX sem - userlevel

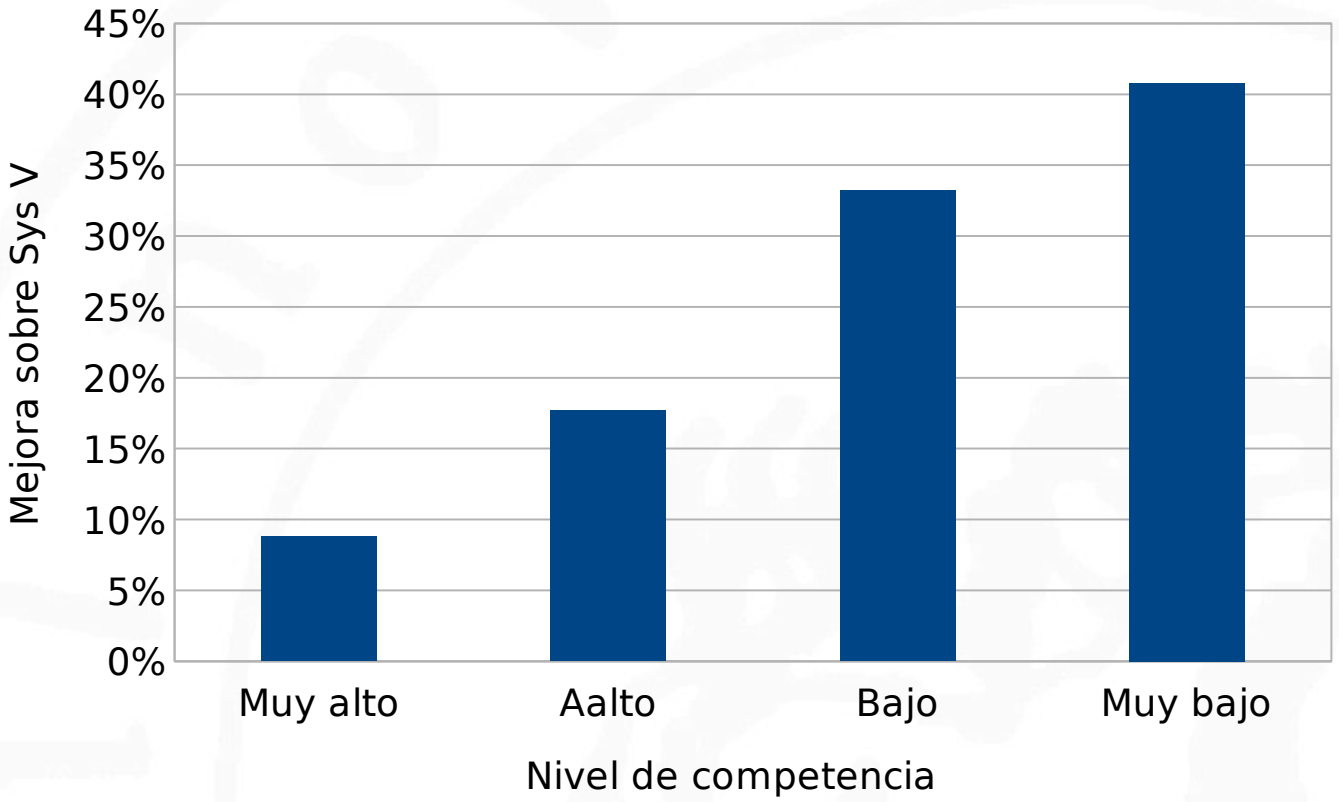
```
int sem_post(sem_t *sem) {
  n = atomic_increment(&sem->value);
  if (sem->nwaiters > 0) {
    sys_futex(sem, FUTEX_WAKE, 1);
  }
}
```

Implementación POSIX sem - kernel

- No mantiene ninguna estructura permanente para identificar el FUTEX
  - Lo identifica por su dirección física de memoria
- Los procesos bloqueados se mantienen en una estructura hash table
- Los procesos en el kernel son desbloqueados en forma FIFO
  - ...pero esto no alcanza para garantizar orden

Implementación POSIX sem

- 2 procesos compitiendo



Referencias

- Kerrisk, Michael. The Linux programming interface. No Starch Press, 2010.
- Franke, Hubertus, Rusty Russell, and Matthew Kirkwood. "Fuss, futexes and furwocks: Fast userlevel locking in linux." AUUG Conference Proceedings. AUUG, Inc., 2002.
- Drepper, Ulrich. "Futexes Are Tricky." (2004).